CS 109: Problem Set 6 Coding Problems

Machine Learning Classifier Assignment

Please read this document carefully while you implement the Naive Bayes and Logistic Regression classifiers for this assignment. Good luck, and have fun!

Problem 4: Naive Bayes Classifier

Implement a Naive Bayes Classifier in `naive_bayes.py` .

Part 4(a): `fit()`

In `naive_bayes.py` , implement the `fit()` method where you see `YOUR CODE HERE` .

Parameters:

- `train_features` : a 2D numpy array of shape n x d where rows represent the n samples and columns represent the d features
- `train_labels` : a 1D numpy array of shape n where each value corresponds to the correct label for a respective row in `train_features`

Instruction:

- Set the instance variable `self.label_counts` to a Python dictionary which maps labels to their frequencies. There will be EXACTLY two keys corresponding to the two labels of 0 and 1.
  - The key is a number label (0 or 1 from our data)
  - The value is a number frequency
  - For example:
    > Consider `train_labels = [0, 0, 0, 1, 0, 1]`
    > The label 0 occured 4 times and the label 1 occured 2 times. This results in `self.label_counts` being set to `{0: 4, 1: 2}`
- Set the instance variable `self.feature_counts` to a Python dictionary which maps tuples of feature/label combinations to their frequencies.
  - The key is a tuple (NOT list) of 3 ints: (column number, feature value, label)
  - The value is a number: frequency
- For example:
  > Consider a `self.feature_counts` of `{(3, 1, 0): 17, (7, 0, 1): 2, ...}`
  > Column 3 with value 1 and output label 0 occurred 17 times in training data
  > Column 7 with value 0 and output label 1 occurred 2 times in training data
  > (there will be many others)

Hints:

- Using `self.feature_counts[key]` will raise a `KeyError` if `key` does not exist in `self.feature_counts`. A better alternative is `self.feature_counts.get(key, 0)` which will default to 0 if `key` is absent.
- No key in `self.feature_counts` should map to a value of 0; if a key does not occur, then it should not be included in the dictionary.
- These counts are NOT affected by the value of `self.use_mle`.
- Remember you are not returning any values; rather, you will set the values of the instance variables `self.label_counts` and `self.feature_counts`.
- Remember that all the data is zeros and ones. This means that in the tuple key of `self.feature_counts`, the range of the second value is [0 - 1], and the range of the third value is [0 - 1]. The range of the first value is [0 - (number of columns minus one)]

Part 4(b): `predict()`

In `naive_bayes.py`, implement the `predict()` method where you see `YOUR CODE HERE`.

Parameters:

- `test_features`: a 2D numpy array of shape n x d where rows represent the n samples and columns represent the d features

Instruction:

- Given `n` rows of features, return a numpy array `preds` of length `n` (initialized for you to all 0's) so that `preds[i]` contains the classifier's prediction for sample `i`. This involves looping through each row in `test_features`, computing the unnormalized probability that the given row should be labeled 0, and computing the unnormalized probability that the row is a 1. Then, assign `preds[i]` to 0 if 0's probability is greater than or equal to 1's probability, and set to 1 otherwise.

Hints:

- You will be making use of the counts that you computed in the fit method to compute the probability of each row.
- The boolean variable `self.use_mle` affects whether or not to use Laplace add-one smoothing.
- Remember you are assigning `preds[i]` to 0 if its unnormalized probability is greater than 1's unnormalized probability, NOT if it is greater than 0.5. Also, note the probability for 0 and the probability for 1 will not necessarily add to 1.
- You can use log likelihood or regular likelihood. Both achieve the exact same results.

Problem 5: Inference with Naive Bayes

Perform inference with the Naive Bayes Classifier in `questions.py`.

Part 5(a)

(COMPLETED FOR YOU) In `questions.py`, implement `question_nb_a()`.

Question:

- You are given an untrained Naive Bayes Classifier initialized to use MLE, as well as training features and labels for the Netflix dataset.
- What is the prior probability that a user rates highly the film "Love Actually" (corresponding to the output label) unconditional on any preference data from other films?

Instruction:

- Compute the classifier's estimate for `P(Y = 1)` using its instance variables.
- Return a single floating point number for the probability.
- This function has been completed for you. Pay attention to how we use the parameters because other functions we ask you to complete will have a similar set of parameters (but with different values depending on the problem).

Parameters:

- `clf` : In this case we use Naive Bayes, but Logistic Regression would be automatically supplied if the problem calls for it. The classifier has already been initialized with its hyperparameters (in this case `use_mle=False` , but it would use `learning_rate` and `max_steps` if the problem called for Logistic Regression).
- `train_features` : a 2D numpy array of shape n x d where rows represent the n samples and columns represent the d features. For this question, it represents the data from the Netflix dataset.
- `train_labels` : a 1D numpy array of shape n where each value corresponds to the correct label for a respective row in `train_features` . For this question, it represents the data from the Netflix dataset.

Part 5(b)

In `questions.py` , implement `question_nb_b()` where you see `YOUR CODE HERE` .

Question:

- You are given an untrained Naive Bayes Classifier initialized to use MLE, as well as training features and labels for the ancestry dataset.
- For each Single Nucleotide Polymorphism, predict the probability that it matches the human reference genome given that the DNA is from a person of East Asian descent.

Instruction:

- For all values of `i` , compute the classifiers estimate of `P(X_i = 1 | Y = 0)` .
- Return a numpy array of floats of the form:

```
[ P(X_0 = 1 | Y = 0),  P(X_1 = 1 | Y = 0),  P(X_2 = 1 | Y = 0), ...]
```

Part 5(c)

In `questions.py` , implement `question_nb_c()` where you see `YOUR CODE HERE` .

Question:

- You are given an untrained Naive Bayes Classifier initialized to use MLE, as well as training features and labels for the ancestry dataset.
- For each Single Nucleotide Polymorphism, predict the probability that it matches the human reference genome given that the DNA is from a person of Ad Mixed American descent.

Instruction:

- For all values of i, what is your estimate P(X_i = 1 | Y = 1)?
- Return a numpy array of floats of the form

```
[ P(X_0 = 1 | Y = 1),  P(X_1 = 1 | Y = 1),  P(X_2 = 1 | Y = 1), ...]
```

Part 5(d)

In `questions.py` , implement `question_nb_d()` where you see `YOUR CODE HERE` .

Question:

- You are given an untrained Naive Bayes Classifier initialized to use MLE, as well as training features and labels for the Netflix dataset.
- Which five movies are the most indicative that a user will like the film "Love Actually" (which corresponds to Y = 1).

Instruction:

- For which five `i` is the ratio `P(Y = 1 | X_i = 1) / P(Y = 1 | X_i = 0)` the largest?
- Return a numpy array of the 5 indices. The elements in the output need to be sorted ordinally (NOT by importance).
- For example:
  > Assume that the result was [9,4,18,17,12] sorted by importance.
  > You should return [4,9,12,17,18] sorted by ordinal.

Hints:

- Make sure you treat each i as zero indexed.
- There are 19 columns, so your answers should be (0-18) inclusive.
- You may find `np.sort()` helpful.

Problem 6: Logistic Regression Classifier

Implement a Logistic Regression Classifier in `logistic_regression.py` .

Part 6(a): `fit()`

In `logistic_regression.py` , implement the `fit()` method where you see `YOUR CODE HERE` .

Parameters:

- `train_features` : a 2D numpy array of shape n x d where rows represent the n samples and columns represent the d features
- `train_labels` : a 1D numpy array of shape n where each value corresponds to the correct label for a respective row in `train_features`

Instruction:

- Update the theta vector by performing gradient ascent.
- You should iterate `self.max_steps` times, and for every gradient you compute, you should scale it by `self.learning_rate` .

Hints:

- You should make use of the provided sigmoid function.
- Performance here is important. First, implement the basic algorithm with many loops. Second, try to find any repeated operations you are performing in the innermost loop, and compute them just once before that loop. Third, make use of `numpy` array operations. (e.g. use `arr1 + arr2` instead of looping over individual elements).
- This takes some time to execute on the Netflix dataset depending on your implementation. You should expect milliseconds with one loop and a few minutes with two loops. With three loops, it can take anywhere from a few minutes to over an hour. Usually, the problem is too intractable to grade with three loops.
- The vectors `train_labels` and `theta` each have only a single axis (of size `n` or `d` respectively). If it helps you to think in terms of a row vector (with two axes, the *first* axes of length one), you can use `arr.reshape(1, arr.size)` . If it helps you to think in terms of a column vector (with two axes, the *second* axes of length one), you can use `arr.reshape(arr.size, 1)` . Both these reshape operations are always constant time (i.e. O(1)) when used on arrays of 1 axis. Note that reshaping is not necessary for a correct implementation.
- You can do matrix multiplication in `numpy` using `np.matmul` , or by using the `@` symbol. That is, `A @ B` will return the matrix-matrix product AB, `A @ x` will return the matrix-vector product Ax, and `x @ y` will return the dot product

xy (these all assume that the dimensions align correctly). For details on the subtleties of matrix multiplication in `numpy`, see the "Notes" in the docs here.
- In most implementations of this function, elementwise operations (e.g. `+`, `-`, `*`, `/`) are always between arrays of the same dimensions and behave as you would expect. If you are curious about the behavior of elementwise operations for arrays with *different* dimensions, see numpy broadcasting. Note that broadcasting does not apply to *matrix multiplication* because it is not considered an *elementwise* operation.
- If you are failing tests but the values are close to the correct values, it is much more likely that there is a minor bug in your code than a numerical precision issue. Ensure that you are updating the entire `theta` vector before recomputing the gradient (do NOT update one index and recompute the whole gradient each time). This is not a common problem, but may cause results that look only approximately correct.

Part 6(b): `predict()`

In `logistic_regression.py`, implement the `predict()` method where you see `YOUR CODE HERE`.

Parameters:

- `test_features`: a 2D numpy array of shape n x d where rows represent the n samples and columns represent the d features

Instruction:

- Given `n` rows of features, return a numpy array `preds` of length `n` (initialized for you to all 0's) so that `preds[i]` contains the classifier's prediction for sample `i`. This involves applying the weights we learned in the `fit()` method.

Hints:

- The remaining functionality can be written in few lines if you take advantage of matrix multiplication, but this is not required.

Problem 7: Inference with Logistic Regression

Perform inference with the Logistic Regression Classifier in `questions.py`.

Part 7(a)

In `questions.py`, implement `question_lr_a()` where you see `YOUR CODE HERE`.

Question:

- You are given an untrained Logistic Regression Classifier, as well as training features and labels for the heart dataset.
- Which five regions of interest in the heart scan are the strongest predictors that patient has heart disease?

Instruction:

- Return the indices of the five most important weights.
- The value determines the importance, NOT the absolute value (e.g. -2 is less important than +1).
- Return a numpy array of the 5 indices. The elements in the output need to be sorted ordinally (NOT by importance).
- For example:
  > Assume that the result was [9,4,18,17,12] sorted by importance.
  > You should return [4,9,12,17,18] sorted by ordinal.

Hints:

- Make sure you treat each i as zero indexed.
- There are 22 columns and 1 bias term, therefore 23 weights. Your answers should be in the bounds [0-22] inclusive.

- Consider the bias weight as "feature 0". For our purposes, no special consideration should be made for it.
- You may find `np.sort()` helpful.

Part 7(b)

In `questions.py` , implement `question_lr_b()` where you see `YOUR CODE HERE` .

Question:

- You are given an untrained Logistic Regression Classifier, training features & labels, as well as test features & labels for the heart dataset.
- What percentage of the time is a classification of heart disease actually heart disease when evaluated on the TEST set?

Instruction:

- This metric is known as `Precision` . It is the ratio of true positives to [true positives plus false positives] (i.e. the ratio of correctly labeled 1's to all 1's that the classifier labeled).
- What is the `precision` of the classifier evaluated on the heart TEST set?
- Return a floating point number.

Hints:

- You should make use of the `predict()` method of the classifier

Part 7(c)

In `questions.py` , implement `question_lr_c()` where you see `YOUR CODE HERE` .

Question:

- You are given an untrained Logistic Regression Classifier, training features & labels, as well as test features & labels for the ancestry dataset.
- What percentage of Ad Mixed Americans are recognized as such by the classifier when evaluated on the TEST set?

Instruction:

- This metric is known as `Recall` . It is the ratio of true positives to [true positives plus false negatives] (i.e. the ratio of correctly labeled 1's to all actual 1's in the labeled data)
- What is the `Recall` of the classifier evaluated on the heart TEST set?
- Return a floating point number.

Hints:

- You should make use of the `predict()` method of the classifier.

Testing

- **Ensure you are using python version >= 3.6.**
- **Ensure that you have `numpy` installed for the version of python you are using.**
- **Do NOT change the folder structure, rename files, or delete files.**
- **Tests that output as `HIDDEN` do not tell you if they passed or failed when you run locally. They merely allow you to observe their output locally.**
- **The autograder on Gradescope tells you whether your answer is correct, though it won't tell you what the correct answer is.**

Command line

In these commands, replace `python` with `python3` if needed for your system.

Test your classifiers:

```
python test_classifiers.py
```

Test your questions:

```
python test_questions.py
```

Test individual functions by listing them:

```
python test_classifiers.py predict_bayes_mle_simple fit_logistic_heart
```

From python

Note that you don't need to pass in the classifiers or datasets as parameters to the test. The environment takes care of these parameters for you. See below.

```
>>> import test_classifiers
>>> test_classifiers.predict_bayes_mle_ancestry()
```

Silence output:

```
>>> import test_classifiers
>>> test_classifiers.predict_bayes_mle_ancestry(display_question=False)
```

Play around with your classifiers:

```
>>> import numpy as np
>>> from naive_bayes import NaiveBayes
>>> n = NaiveBayes(use_mle=True)
>>> n.fit(np.array([[0,0,0,0], [1,1,1,1], [1,0,1,0]]), np.array([1, 0, 1]))
>>> n.predict(np.array([[0,1,0,1], [1,0,1,0]]))
array([0, 1], dtype=uint8)
```

Acknowledgments

Authors:

- Tim Gianitsos - classifiers, solution code, code questions, autograder, instructions
- Anand Shankar - comments, spec, autograder
- Alex Tsun - comments, repo maintenance
- Lisa Yan - spec, questions
- Chris Piech - questions and datasets (compiled prior to Autumn 2019-2020)
- Mehran Sahami - questions (compiled prior to Autumn 2019-2020)